SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 983–997 (1981)

# Document Preparation Systems

HASSE HANSSON AND JØRGEN STEENSGAARD-MADSEN
*Institute of Datalogy, University of Copenhagen, Sigurdsgade 41, DK-2200 Copenhagen N, Denmark*

**SUMMARY**

**This paper gives an introduction to programs used to prepare the layout of a text for publication. It presents a suitable terminology, independent of any particular systems but in no way complete, for the description of documents. The basic characteristics of phototypesetting equipment are described. Finally, highlights of a particular system are given. Very little is said on the implementation of that system.**

KEY WORDS Document preparation system Phototypesetter Layout Text handling

## INTRODUCTION

The painful process of transforming a manuscript to a finally published document can in many ways be eased by use of computers. As mini- and microcomputers become cheaper, such applications will probably become widespread. This paper is written on the basis of experience gained from the implementation of a program designed to ease the way from a manuscript to a phototypeset original for final printing. A design goal for that program was ease of use even at the expense of neglecting some facilities offered by the equipment. The implementation is for a UNIVAC 1100 machine, which is not exactly a microcomputer, but nothing in the system design requires a large machine.

The section on terminology introduces some common terms independent of any particular program or hardware. Terms, which are dependent on either, will be described elsewhere.

In the section on phototypesetters the basic characteristics of the hardware are described and so are the basic capabilities of such equipment.

PHOTODOC is the name of a program which can be used to control a specific phototypesetter. Highlights of the use of this program are given. Use of the program is described on different 'levels' corresponding to the increased skill required of the user. Examples are given to illustrate the form of manuscript, proof listing and final result.

Only a few comments will be made on the implementation. Besides the overall structure and a reference to the programming techniques used, the performance characteristics are described.

Text handling as described requires original typing, subsequent editing and preparation of the text for phototypesetting. Programs used in connection with typing and editing are not described as it is our conviction that whatever the standard utility programs normally provided with a computer system, they should be used for these purposes.

0038–0644/81/090983–15$01.50
© 1981 by John Wiley & Sons, Ltd.
*Received 6 February 1981*

## TERMINOLOGY

This section describes a number of concepts used in later sections to describe a document preparation system. The purpose is to present concepts that are independent of any particular system, so a few of the concepts introduced are not supported by the system to be described later on.

When talking about a text considered in the abstract, we presume as basic concepts: chapters, sections, paragraphs, words and characters. A text may, however, be represented in ways dependent on the medium on which the text may eventually be published. A representation forces us to accept another structure, described by another set of concepts: pages, lines, physical dimensions, typefounts, etc. These concepts are to a large degree equipment dependent, and they must be combined with the basic concepts mentioned earlier to form the set of concepts necessary when describing document preparation systems.

Let us consider typewriter-like automatic equipment as an example of the representation of text. Such equipment is fairly simple, as the only structure that may vary is the line, which is divided into a number of physically identical positions. Each position may be identified in several ways, all dependent on some enumeration. A typewriter operated by a person is, in contrast, a much more flexible tool allowing different spacing between lines, maybe different typefounts (typeballs), and rather liberal positioning for, say, overprinting.

A *document preparation system* is a program, which produces a representation of a text dependent on a particular equipment to be used.

A *running text* is a sequence of running paragraphs.

A *running paragraph* of a text consists of a sequence of words.

The *basic transformation* is the fundamental operation of document preparation. Provided a running text in some representation and the hardware characteristics of some desired representation, the basic transformation will represent the text as desired, keeping the division into running paragraphs. The line structure imposed by the desired representation will generally differ from the line structure in the source text.

To clarify the basic transformation, let us consider the familiar example of typing a letter from a longhand original. It includes a basic transformation of the main part of the letter, the division into lines being determined by the typewriter, the paper and some office rules as to margin settings and separation of paragraphs.

Document preparation systems will as a minimum provide the basic transformation. To enable the system to recognize a running paragraph, input must be typed according to some characteristics for the representation of running text. A desirable goal seems to be characteristics close to those used in letters or books, i.e. separation of paragraphs by blank lines or indented first lines. In this way typing a source text for a document preparation system may be equivalent to the typing of an ordinary letter. Some systems, however, require more explicit information (called *commands*) to separate running paragraphs. The advantage of a simple document preparation system is that even a sloppy text may be neatly printed. A source text may consequently be maintained by any editor program.

It is often a great help in layout to use sketches of the desired appearance of the final document. We will introduce some established techniques from that field in order to illustrate some of the concepts introduced. A running paragraph will be illustrated by a sequence of wavy lines. A page in a book or in a letter can thus be sketched as shown in Figure 1.

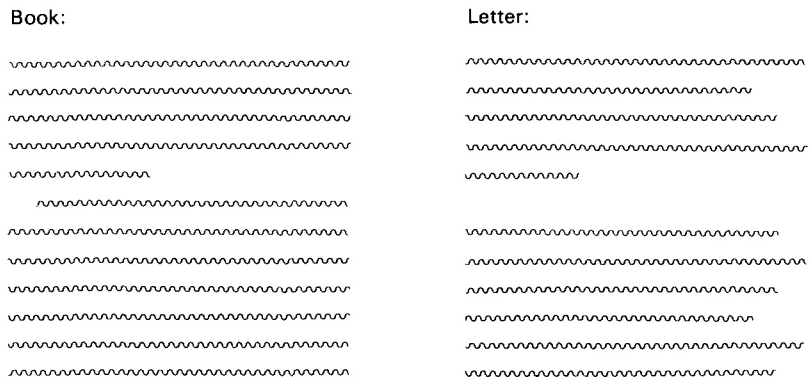


*Figure 1*

To begin a paragraph in a book with an indented line is common practice, and usually the lines, except the last of a paragraph, extend to a common right margin. In a letter, blank lines may separate paragraphs and the lines usually do not extend to a common right margin. Both layouts illustrate that a running paragraph adheres to a left margin. These simple observations reflect the basic differences in the equipment which will be described in the next section. More sophisticated layouts are commonly seen in newspapers, but a description of such layouts is beyond the scope of this paper.

We will introduce a few terms for the description of a layout. Until now only running text has been illustrated (Figure 1). We will use solid lines to denote text that is not part of a running paragraph. As an example take the page shown in Figure 2. The broken lines are inserted in the figure for reference only, e.g. lines 1 and 4 suggest that the lines of the first and the last running paragraphs extend to the *left* and *right limits*. In between these paragraphs we recognize three others with a different set of common limits. Further we note that text appears to the left of *the left flush line* (line 2) for the three middle paragraphs. The space between the left limit (line 1) and the left flush line (line 2) will be called *display space* and texts appearing there *display texts*. We will assume that a display

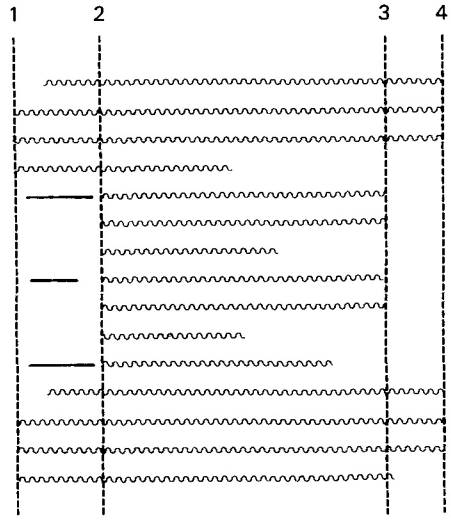


*Figure 2*

text always fits into one line of the display space, even in source texts. We will not use any particular term for the space to the right of the *right flush line* (line 3).

The layout of the pages in a book may be even more sophisticated as shown in Figure 3. A left-hand page and its corresponding right-hand page are called an *opening*. The margin line is a fixed extreme limit (line 0 and $5_R$). The space between line 0 and 1 and the space between line $4_R$ and $5_R$, are called the *marginals*, and text appearing here is called *margin text*. It may be a text which extends over a few lines, but we will not expect it to extend to a common flush position. The space between line 4 and $1_R$ (towards the back of the book) will not be named and we will assume that no text appears here. Note that a left-hand and a right-hand page are not symmetrical as the display text always appears to the left of a running paragraph. The task of distinguishing between left-hand and right-hand pages should be left to an automatic document preparation system.

While introducing the last common terms let us concentrate on a left-hand page. A page in a book may have a complete layout such as shown in Figure 4. The space between the lines A and B is called the *heading* and the space between the lines C and D is called the *footing*. The heading and the footing may contain one line of text, usually divided into three subfields: a left-hand, a centre and a right-hand part.

It is not reasonable to expect a complex desired layout to be expressed by a corresponding layout of the source text. Instead, the desired layout can be described by explicit information, *commands*, interspersed with the source text. If we assume commands to appear on separate source lines, we are lead to a classification of lines in the source text. Such a classification can be represented in several ways. The point to be made here is that rules for interpretation of the lines may depend on the class, e.g. a chapter heading may be classified separately, and actions like advancing to a new page may be associated with such lines.

## PHOTOTYPESETTER HARDWARE

To describe the characteristics of a phototypesetter let us consider a typewritten page and a page composed by a phototypesetter and discover their basic differences. Then we will describe the phototypesetter hardware that imposes these characteristics.

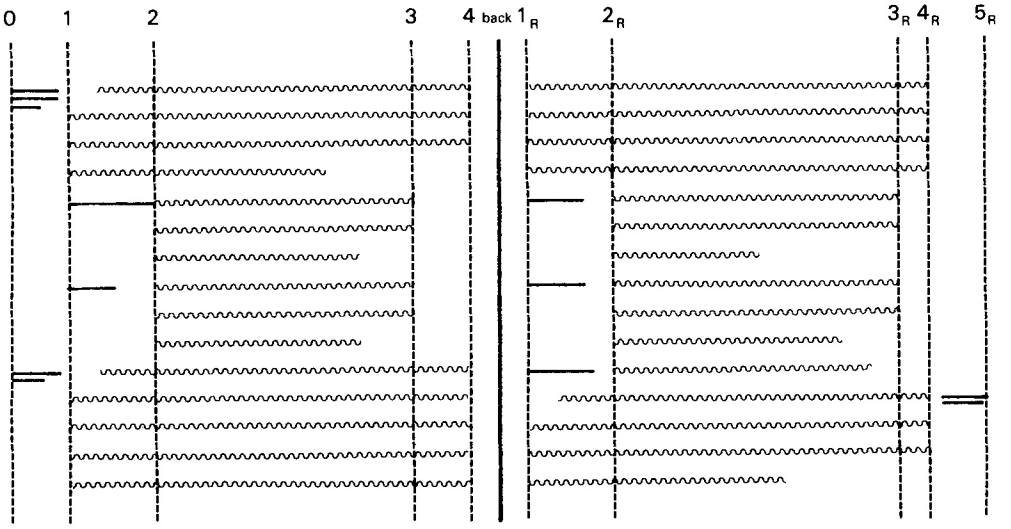


*Figure 3*

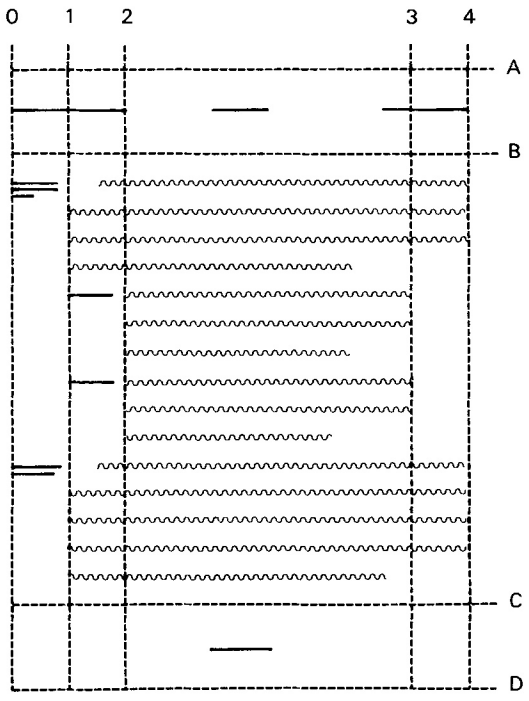


*Figure 4*

A line of a typewritten text consists of a number of characters in physically identical positions. The words are separated by an empty position also called a blank character in common jargon. As the lines contain a different number of characters, they will not have a common right limit. A line of a phototypeset text also consists of a number of characters, but the separation of the characters are determined by the width of the graphic representation. The words are separated by a blank space of a width that is equal within a line but may be different from one line to another. This blank space is called a *space band*, and its length is adjusted, so that a line extends to a common right limit. Such lines are called *justified lines*.

When typing, we use a rather small character set, and only advanced typewriters facilitate use of different character sets, e.g. by replacement of the typeball. A phototypesetter offers a fixed but rather large set of graphic symbols. Furthermore, the characters may be printed in different sizes, so that a chapter heading may be printed in large types and a footnote in small types.

We also note that the line spacing may be chosen as desired in a phototypeset text, while the line spacing in a typed text belongs to a very limited set of fixed line spacings.

In summary the phototypesetter differs from a typewriter in the following respects:

(a) Lines may be jusitified by expanding the space bands.
(b) A large set of characters may be used.
(c) The characters may be typeset in different sizes.
(d) Any desired line spacing may be used.

The remainder of this section will be used to describe the phototypesetter for which the PHOTODOC system has been implemented. The principle used in this second

generation phototypesetter (PHOTON 737 Econosetter) is only one of several possible. Our present phototypesetter will soon be replaced by a third generation machine (Compugraphic 8600), which is based on a quite different principle.

The function of the phototypesetter is illustrated in Figure 5 corresponding to a view from above. The *character disc* is a fast-rotating glass disc with negative pictures of all the characters. The *mirror carriage* may be moved horizontally from side to side (up and down in the figure) so the light beam can be moved. The *lens* may be selected from a fixed set to expose the characters in different sizes. Finally, the *photosensitive paper* may be moved vertically (out of the figure).

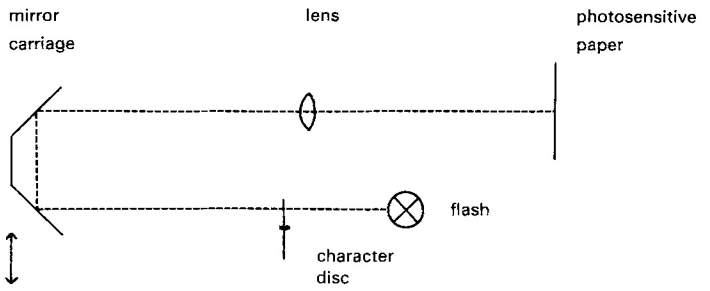


*Figure 5*

Let us assume that the phototypesetter has read a line from the input stream. A computer integrated in the phototypesetter calculates the difference between the desired line length and the minimum length obtained by assigning a fixed minimum width to the space bands. The difference will be divided by the number of space bands in the line giving the *expansion quotient*. On the character disc we find the negative picture of the first character. When this picture is outside the flash it fires, and the character will be exposed onto the photosensitive paper via the mirror carriage and the lens. Before exposing the next following character, the mirror carriage will be moved according to the width of the character printed. If the character is a space band, no flashing occurs, but the mirror carriage will be moved as if the space band had a width equal to the sum of the fixed minimum width and the expansion quotient.

The synchronization and the calculations are carried out by a minicomputer, which is an integrated part of the phototypesetter. The minicomputer combines the primitive hardware facilities in different ways, and it offers these combinations together with some storing facilities to the user as the set of *function codes*. This means that the minicomputer works as an interface between the user and the hardware, and that controlling the phototypesetter is effected by inserting the function codes in the input stream. An example of a simple function code is the 'End-of-line' code. This code causes the line to be exposed, the photosensitive paper to be moved upward and the mirror carriage to be moved to its leftmost position ready for a new line.

To a human user, however, the function codes seem very primitive and much too machine-oriented. Writing a text and inserting all the necessary function codes would be a very time-consuming task and would demand a large knowledge of typography. Thus a document preparation system, acting as an interface between the phototypesetter hardware and the human user, allows the latter to operate on a higher problem-oriented level.

## THE PHOTODOC SYSTEM

This section describes the PHOTODOC system, which is a document preparation system running on a UNIVAC 1100 machine at the computing centre of the University of Copenhagen (RECKU). The system will be described on three different levels reflecting the different skills required by the user.

### Terminology and formats of the input lines

For historical reasons PHOTODOC has its input conventions determined by the fact that they should preferably be a simple extension of those of a typewriter oriented document preparation system. Thus the PHOTODOC system requires that the text be entered using a typewriter-like medium. This means that we may take advantage of the characteristics of a typewriter, such as the division of a line into physically identical positions. In order to distinguish the text from the commands we will interprete all lines having a non-blank character in their first position as commands, and all other lines, *text lines*, will be interpreted as plain text.

These rules make it easy to distinguish the commands from the text when reading the source text. Furthermore, the need of reserving a special character to mark the beginning of a command has become superfluous.

### The introduction level

On this level the user simply enters running text, as if he desired to use an ordinary typewriter for the final document. This means that if he wants to obtain a text composed by a phototypesetter, *the only thing he has to do is to write the text*. The text will pass the basic transformation of the system, but will not be otherwise affected. Paragraphs are separated by blank lines or initiated by an *indented line*, i.e. a text line that has blanks in the first two typing positions.

The user may take advantage of the basic transformation and start a new sentence on a new line, which eases a later editing of the text. Likewise editing the text will be easy, as the right limit of the lines in the source text is unimportant.

The phototypesetter makes it possible to write the characters in different typefaces. This facility is often used to write new terms in italics or important words in boldface. By combining the textline with an optional *typeface line*, which contains information about the desired typefaces, the user may select other typefaces apart from the plain one. A typeface line is distinguished from other lines by a + in its first position. The typeface lines consists of characters selecting the desired typeface of the above written character. They are chosen in conformity with the Danish standard as follows:

(blank) under plain types
~ (tilde) under *italicized types*
- (hyphen) under **semi-bold types**
= (equal sign) under **bold types**

This approach is in correspondence with normal manuscript writing, and it does not make the source text difficult to read.

The lines produced by the basic transformation must be divided into a number of pages. This will automatically be done by the system, and the pages will, apart from the text, contain a page number incremented by the system. The division into pages will often cause a running paragraph to be broken between two pages. The system will in this case make the break, so that no line beginning a running paragraph will be left alone on a page, meaning that the so-called *widows* are avoided.

When going from a manuscript to a final paper, one often needs a preliminary copy for proofreading. Therefore the PHOTODOC system has been designed to produce three different kinds of output: a *proof listing*, a *justified source*, and the *final typesetting*.

The proof listing is written on a conventional lineprinter, because it is a fast and an inexpensive output medium. The proof listing is a page-proof, not a galley proof, which allows you to check the layout of the pages in addition to the text itself. However, as the graphic facilities of a lineprinter are very limited compared to a phototypesetter, the proof list must contain additional information. One example related to the introduction level is the typeface information, which looks almost as a typeface line in the input. In Figure 6 you will see a proof listing and its typeset equivalent.

The justification has one disadvantage: When you proofread and find an erroneous word in the text, it may be difficult to locate the corresponding word in the source text. To eliminate this drawback PHOTODOC may produce a justified source, which is identical with the original source except that all text lines are written according to the justification. The proof list will in this case contain a *line number* to the left of each line, and in this way the erroneous word may immediately be found and corrected.

## The simple application level

Commonly users have an idea of the typographical results that may be obtained by phototypesetting and document preparation systems. These include the use of various sizes of text, indentations of parts of a text, automatic generation of table of contents and a lot more. Such results are achieved by the use of commands in PHOTODOC. Basically, commands may only appear outside running text — i.e. commands take effect immediately prior to the opening of a new paragraph. This means, for instance, that it becomes very hard to describe a line that contains characters of different sizes, and that it is impossible to change the length of a line in the middle of a paragraph.

Most commands take parameters that may be numbers, mnemonics or text. Consider for example the command INDENT, which takes two parameters, one mnemonic and one real number. It is used to situate the left and right flush lines relative to the corresponding limit lines. The mnemonics allowed are: RIGHT, LEFT and BOTH with obvious reference to the flush lines. Thus the command

```
INDENT BOTH, 15.5 mm
```

will indent both flush lines 15·5 mm towards the centre of the page.

The division of the text into chapters, sections, subsections etc. is controlled by the commands 1,2, ..., 9 (here stated in corresponding sequence). Each takes one parameter: A text that becomes the heading of the succeeding running text. Unless explicitly switched off, an automatic mechanism takes care of proper separation by blank lines around such headings and ensures that it will be typed in semi-boldface. Furthermore, the use of these commands implies the generation of a corresponding table of contents.

The size of the types to be used for subsequent text is controlled by use of the commands GIANT, LARGE, MEDIUM and SMALL. With more flexible hardware these will probably collapse into one parameterized command. They are mentioned here because the use of various sizes causes a problem in the proof list. As proof lists are normally written by a simple lineprinter, various sizes are not available for them. It has proved satisfactory to encode the size information as one character preceeding each line in the proof list. The most common size is MEDIUM, which is encoded by a dot, whereas the

```
. Inddata                                PHOTODOC

. SETCTR
  ------
. PHOTODOC  ajourfører  automatisk
  de tællere, der bestemmer afsnits-,
  fodnote- og sidenumre. Undertiden
. er det ønskeligt at gribe ind i denne
. automatik; dette vil fx være tilfæl-
. det, hvis man redigerer en tekst i
. flere selvstændige elementer. I så-
. danne  tilfælde  kan  kommandoen
. SETCTR  anvendes til at give tæl-
. leren en  givet  værdi.  Kommando-
. formatet er:
.
.          SETCTR tællertype, n
.
. hvor  n er et heltal og tællertype kan
. være:
. 1-9:Kapitel-  og  afsnitstællere.
.     Tælleren sættes til  n  -  1,  så-
.     ledes  at  den  ved  næste kapitel
.     eller afsnit antager værdien  n.
.     Se  i  øvrigt afsnit 2.4.8 for en
.     nærmere  beskrivelse  af  kapi-
.     teloverskrifter  og  afsnitsover-
.     skrifter.
. n:  Fodnotetæller.  Næste  fodnote
.     får nummer  n.
.
. 48
```

```
. Manual                                 Inddata

. p:  Sidetæller.  Næste  side  tilde-
.     les værdien  n + 1.
. t:  Sidenummeret  på  indholds-
.     fortegnelsens  første  side.  Nor-
.     malt  sættes  sidenummeret  til
.     1;  men  med  kommandoen
.     SETCTR  T  kan  sidenummeret
.     i stedet sættes til  n.
.
. SYM
  ---
. Kommandoen  er  især  velegnet  ved
. interaktiv  kørsel  ("demand",  "on-
. line"),  hvor  den  anvendes  til  at
. dirigere  korrekturteksten  til  en an-
. den  uddataenhed  end  den  terminal,
. der  arbejdes  ved.  Kommandoen ef-
. terfølges af navnet på den alterna-
. tive  uddataenhed.  Fx  vil  komman-
. doen
.
. SYM PR2
.
. dirigere korrekturteksten til linie-
. skriveren  i  selvbetjeningsrummet.
. Det er altså ikke nødvendigt at op-
. rette en fil og senere sende den til
. uddataenheden  ved  hjælp  af  eks-
.
.                                        49
```



**SETCTR**

PHOTODOC ajourfører automatisk de tællere, der bestemmer afsnits-, fodnote- og sidenumre. Undertiden er det ønskeligt at gribe ind i denne automatik; dette vil fx være tilfældet, hvis man redigerer en tekst i flere selvstændige elementer. I sådanne tilfælde kan kommandoen SETCTR anvendes til at give tælleren en givet værdi. Kommandoformatet er:

SETCTR *tællertype, n*

hvor *n* er et heltal og *tællertype* kan være:

1-9: Kapitel- og afsnitstællere. Tælleren sættes til *n* - 1, således at den ved næste kapitel eller afsnit antager værdien *n*. Se i øvrigt afsnit 2.4.8 for en nærmere beskrivelse af kapiteloverskrifter og afsnitsoverskrifter.

n: Fodnotetæller. Næste fodnote får nummer *n*.





p: Sidetæller. Næste side tildeles værdien *n* + 1.

t: Sidenummeret på indholdsfortegnelsens første side. Normalt sættes sidenummeret til 1; men med kommandoen SETCTR T kan sidenummeret i stedet sættes til *n*.

**SYM**

Kommandoen er især velegnet ved interaktiv kørsel ("demand", "online"), hvor den anvendes til at dirigere korrekturteksten til en anden uddataenhed end den terminal, der arbejdes ved. Kommandoen efterfølges af navnet på den alternative uddataenhed. Fx vil kommandoen

SYM PR2

dirigere korrekturteksten til linieskriveren i selvbetjeningsrummet. Det er altså ikke nødvendigt at oprette en fil og senere sende den til uddataenheden ved hjælp af eks-



*Figure 6*

others are encoded by the initial letter of the corresponding command. This helps the reader by emphasizing the unusual, and we find this an important property. Furthermore, this encoded information can be removed from the proof list, such that the text may be handed out for comments without such extraneous information.

In addition to the variations in typefaces there exist graphics that are not represented in a standard character code, such as ASCII, but are available for phototypesetting. The command ASCII allows the user of PHOTODOC to redefine the standard correspondence between a typed character and the typeset graphic. This is accomplished by an enumeration of the graphics, initially an extension of the ASCII code. A change in one correspondence is expressed by a command like

ASCII 181 = #

following which the character # in the input text will denote the graphic associated with the number 181 (© in the running version).

It is not unusual to have some display text at the beginning of an indented running text. In PHOTODOC we have inherited a convention from a similar document preparation system, which is not for typesetting however. By this convention the desired layout must be present also in the manuscript (input to the system). Thus it becomes meaningful to distinguish, also in the manuscript, between the left limit and the left flush line. Because the manuscript is supposed to be typed, it will be natural to fix the left flush line by a position number, rather than in millimetres. The PHOTODOC system will then, of course, have to implement some standard transformation between the two measures of the display space. Experience has shown that this can be done satisfactorily. A command of the form COLUMN *n* will have the effect that until repositioned again the left flush line in input will be immediately to the left of the *n*th typing position. Lines indented to the right of the left flush line are considered paragraph openings. Similarly, a text in the (input) display space marks the beginning of a paragraph.

## The advanced level

In this section we describe the possible controls a user has of page-utilization. Obvious parameters are the length of a line and the format of a page. Of importance for readability is the separation between lines in running text. The position of page numbers may vary, and different forms of text for context identification in the heading and footing of a page may be given. Finally, we describe how to set up tables.

The obvious parameters for a page are controlled by the command FORM. Either a standard mnemonic for a common pageformat is used as parameter, or two integers for lengths in millimetres are used.

The spacing between lines is determined automatically when the size of the graphics is determined. This is done according to typographical conventions based on the assumption that a line contains approximately 60 graphics. Longer lines require larger line spacing to prevent a decrease in readability. The command LINESPACE will set the line spacing to the standard value multiplied by the real number given as parameter to the command. The interested user may like to examine a few examples in References 1 and 2.

On every page the top and bottom 25 mm — the heading and footing — are reserved for context identification contained in one line. Distinguishing between left and right pages we find four commands to control the contents of the corresponding four lines:

LEFTHEAD, RIGHTHEAD, LEFTFOOT and RIGHTFOOT. Each of these lines consists of three fields: left, centre, and right. Each field may contain text or a page number, which will be left justified, centred or right justified depending on the field. The following example shows how the commands may be used

```
LEFTHEAD /CHAPTER/ /"PHOTODOC"/
RIGHTHEAD /"Manual"/ /CHAPTER/
```

The slash is used for field separation, CHAPTER is a mnemonic that indicates that the heading of the current chapter is written, and arbitrary text parts are given in quotation marks, which are not part of the resulting line. The centre field is empty in this example. Thus, considering an opening (cf Figure 6) we will find the chapter heading reproduced in the upper corners, and the text PHOTODOC Manual centred (for the entire opening). Page numbers are indicated by the mnemonic PAGENUMBER (or ROMAN if numbering in roman style is desired).

In order to set up tables two things must be taken care of: (1) The final form must be *defined*, and (2) the content of the table must be given. A table consists of *lines* divided into *fields*, which will form a *column* when a sequence of lines belonging to the table are typeset.

The content of a table is correspondingly given in lines divided into fields. In practice the limitation thus imposed has not caused any problems. Usually, the content of a field in the source text is transferred directly to the corresponding field in the final form. However, one exception is when the contents of corresponding fields in a sequence of source lines are considered as lines of one running text. In this case a basic transformation is performed before the content is transferred to the final form.

We distinguish between five kinds of columns, each identified by a one-letter code. Code N denotes a column with running text. The others are distinguished according to the *justification* of the contents in the fields: L denotes left justification, R right justification, C centre justification. The code S denotes fields that are split into a left and a right justified part, and the space between the parts is filled by a leader character (a dot is often used in tables of contents, which may be considered as a table with one column of kind S).

A table definition starts with the command DEFINETABLE taking an identifier as parameter. The identifier is used when the content is given to refer to the final form. Following the command comes a number of *field-descriptions*, each stating the type of the column, the field width in millimetres, the size (stated by the earlier mentioned mnemonics) of the graphics used in that field, and optionally, a fixed content. If a fixed content is given, the column is called *closed*, otherwise it is called *open*. An END command follows the last field description.

Table contents are supplied in lines with as many fields as open columns in the final form. The lines are enclosed by the commands TABLE and END. An identifier is used as parameter in the TABLE command to refer to a table definition. A second parameter determines how fields are recognized in the lines. Recognition can be based on writing positions, which require the source lines to appear as a table themselves, or an explicit separator character, similar to the way fields for headings and footings were recognized.

## PROGRAM STRUCTURE

The PHOTODOC program is a 10,000 line Pascal program, using extended language facilities as described by Steensgaard-Madsen.[3] In this section we will give a view on

the program structure, which adheres to a strict discipline of modularization. The testing of this discipline in practice was a major motivation for both of us when the initial version was developed as a project work during a computer science study. Later, as the program was extended gradually to meet user requirements, the imposed structure has proved to be robust. The program is now used extensively by the users of the computing services center (RECKU) at the University of Copenhagen.

We will give an account of the program structure only as far as to describe the facilities at the introduction level. A few comments at the end of the section will give an idea of the structures used for the more advanced facilities.

The structure can be illustrated by a tree as in Figure 7. The boxes in the tree are subprograms and the lines downwards from a subprogram indicate access rights. The boxes are grouped by their shading, and boxes of different groups belong to different

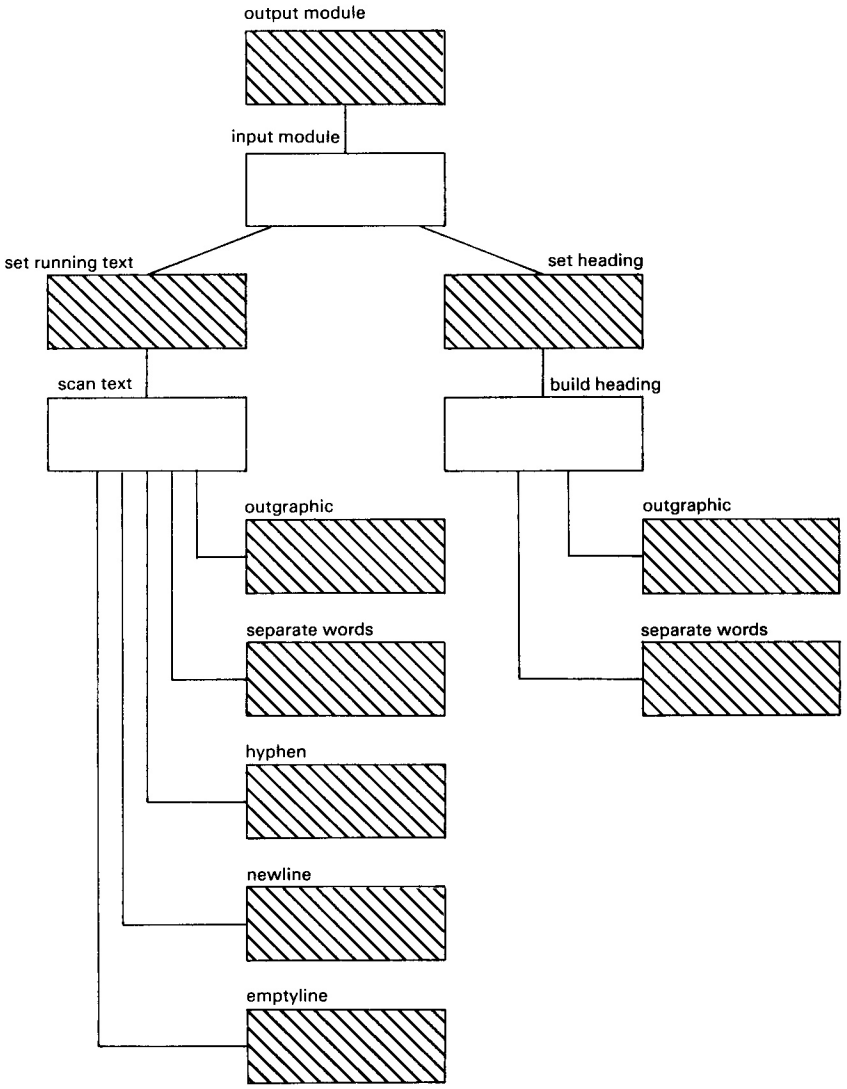


*Figure 7*

contexts in terms of auxiliary variables and subprograms. The separation of contexts is technically achieved by extensive use of the parameter mechanism. Steensgaard-Madsen[4] shows how the technique may be used for the implementation of a more direct means of expressing the separation.

The boxes are identified for easy references. Explanations will follow the structure from top to bottom, separately for the two contexts.

At the top we find the *output module*, which as a text comprises the definitions of all subroutines that are shaded in Figure 7. As a subroutine itself its only responsibility is to initialize variables that are used to reflect the phototypesetting hardware; to activate the *input module* in such a way that access is restricted to the routines *set running text* and *set heading*; and finally, when control returns from input module, do whatever is needed to pass on the data collected to the phototypesetting hardware.

The input module may activate *set running text* several times; during each activation the concerns of *set running text* are:

1. That the received text (cf. *scan text* below) is typeset with fixed values for the positions of the flush lines and for the size of the graphics.
2. That widows (cf. page 989) are avoided.
3. That *scan text* is activated and allowed access as depicted in Figure 7.

The routines accessed by *scan text* are used to communicate the running text from one context to the other, that is from the *input module* to the *output module*. The communicated components of running texts are: (1) graphics, (2) word separations, (3) hyphens recognized at the end of a source line or written as a special character marking a possible hyphenation point (discretionary hyphen), (4) termination of paragraphs and (5) empty lines.

The routine *set heading* is very similar, but simpler in its logic, to *set running text*. The problems of widows has an obvious analogy in so far that headings isolated at the bottom of a page are undesirable, otherwise it is concerned mainly with the typesetting of one line.

Turning now to the logic of the *input module*, we find that it has to define the routines *scan text* and *build heading*; to recognize the division of the source text into commands and running texts; and, for each such part, to activate the appropriate routines *set running text* and *set heading* with access rights as shown in Figure 7.

The routine *scan text* has to decode the source stream of characters into a stream of graphics and pass them on to the *output module* by the means provided as access rights. Furthermore, it is the responsibility of *scan text* to recognize the end of a running text and return without processing the line that caused it. In this way control flow is used as an alternative to complicated communication by status variables throughout the system.

The routine *build heading* is responsible for automatic generation of section numbers and use of semi-bold types in headings.

Additions to the system in terms of more complicated commands should now be fairly obvious. Unless it can be expressed by the existing access rights of the *input module*, new rights have to be defined and provided. When such additions are designed, it must first be decided where a new branch has to be added to the tree in Figure 7. When this has been decided, the branch may be expanded into a subtree in such a way that in the entire resulting tree boxes at alternate levels belong to different modules, and boxes at the same level to the same module.

Experience has shown this overall structure to be very robust. The technique we have used to restrict the access rights is unusual application of the parameter mechanism of block structured languages. When confronted with this technique people tend to question the efficiency of it. For this program it was found not to be the main bottleneck; that turned out to be the I/O.

The modularization technique, which we have illustrated, applies to routines as well as to entire programs. This is not, however, exercised in the current version of PHOTODOC, primarily because our notational technique has not been developed until recently. We believe that the illustration by a *module tree* as used in this paper will be of great value for design as well as for documentation. In this form the approach can be used, although not with full advantage, along with any programming language. Full advantage is gained, when the programming language supports the separation of modules and the restrictions in access rights.

The efficiency of a document preparation system must be good, as it handles a large amount of data (an average novel contains about 250,000 characters). Experience shows that the PHOTODOC system requires less than 1 second* to handle an average page, and that only 20 per cent of this time is CPU-time. About 65 per cent is used for I/O, while the remaining 15 per cent is operating system services. Often you need to output a few, recently corrected pages. In this case PHOTODOC may be instructed to restrict the output to these pages, which will reduce the execution time significantly.

## FINAL REMARKS

We have given an overview of a program used for document preparation. We have described some of the facilities grouped in three according to user sophistication. Throughout we have concentrated on the most detailed form in which a facility may be used. In many cases a facility may be used by simpler means (typically by a command with fewer parameters or by assuming some default usage). Thus, the grouping by three levels is actually too coarse, and we have tried our best to build the system in such a way that a user recognizes a substantial change in his own demands when he realizes the need to learn a new set of commands.

The program offers facilities that have not been mentioned, such as the generation of a subject index, the use of sub- and superscripts and the insertion of footnotes. Another aspect is automatic hyphenation, which is provided for Danish texts only. In this respect the program performs twice as well (i.e. with half the error rate) as any known system. This has been possible only by the professional assistance of a linguist, Bente Maegaard. As a similar assistance has lacked for other languages, and as development of hyphenation programs is a very time-consuming task, we have not yet developed hyphenation routines for other languages.

In a separate section we have described the structure of the program by means of module trees. As a description method this is more recent than the program development. Initially the program was also developed to check out the approach to design and development expressed in part by the module tree. In our own opinion the project has been a success, in so far as the program has grown gradually without damage to its reliability and in so far as it now has a large local user community. One drawback is that the technique used for implementation is considered strange by most programmers, and that the maintenance consequently becomes a serious problem. However,

* Measured in a so-called Standard Unit of Processing for UNIVAC 1100 machines.

the technique is considered strange only because it expresses data abstraction without an explicit syntax for it.

ACKNOWLEDGEMENTS

We are grateful to Benedict Løfstedt, University of Aarhus, for his assistance, when this project was started. We are also grateful to many users, whose constructive criticism has been of great value for the development of PHOTODOC.

REFERENCES

1. John R. Biggs, *Basic Typography*, Faber and Faber, London, 1968.
2. Hasse Hansson, *Fotosættersystemet PHOTODOC*, 4th edn, RECKU, Copenhagen 1980 (in Danish).
3. J. Steensgaard-Madsen, 'Pascal—clarifications and recommended extensions', *Acta Informatica* **12**, 73–94 (1979).
4. J. Steensgaard-Madsen, 'Å statement oriented approach to data abstraction', *TOPLAS*, **3**(1), 1–10 (1981).